DILLON & YUDELL LLP
ATTORNEYS AT LAW

FACSIMILE TRANSMITTAL SHEET

TO: Examiner B. O'Brien	FROM: Brian Russell, Reg. No. 40,796
COMPANY: USPTO	DATE: August 19, 2004
FAX NUMBER: 703.872.9306	TOTAL NO. OF PAGES INCLUDING COVER: 46
PHONE NUMBER:	SENDER'S REFERENCE NUMBER: AT9-99-451
RE: Formal Submission: Appeal Brief	YOUR REFERENCE NUMBER: 09/598,434

☐ URGENT ☐ FOR REVIEW ☐ PLEASE COMMENT ☐ PLEASE REPLY ☐ PLEASE RECYCLE

NOTES/COMMENTS:

The attached Appeal Brief and Terminal Disclaimer are respectfully submitted (in triplicate) for filing in the above-referenced patent application.

B. Russell/vf

8911 N. CAPITAL OF TEXAS HWY., SUITE 2110, AUSTIN, TEXAS 78759
512.343.6116 (V) • 512.343.6446 (F) • DILLONYUDELL.COM

RECEIVED
CENTRAL FAX CENTER
AUG 19 2004

IN THE UNITED STATES PATENT & TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

OFFICIAL

In re Application of: **CHARLES MOORE**	§	Attorney Docket No.: **AT9-99-451**
Serial No.: **09/598,434**	§	Examiner: **CHANG, J.**
Filed: **22 JUNE 2000**	§	
Title: **PROCESSOR AND METHOD OF EXECUTING LOAD INSTRUCTIONS OUT-OF-ORDER HAVING REDUCED HAZARD PENALTY**	§	Group Art Unit: **2154**

APPEAL BRIEF

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

This Brief is submitted in triplicate in support of the Appeal in the above-identified application.

CERTIFICATE OF FACSIMILE TRANSMISSION
37 C.F.R. § 1.8

I hereby certify that this correspondence is being transmitted by facsimile on the below date to the U. S. Patent and Trademark Office, ATTN: Examiner C. Chang, Art Unit 2154, at **703.872.9306**.

Date: 8/18/04 By: Vicky Talyansky

REAL PARTY IN INTEREST

International Business Machines Corporation, the assignee of record as evidenced by the assignment recorded at frame 0052 of reel 010926 of the assignment records of the USPTO, is the real party in interest in the subject Appeal.

RELATED APPEALS AND INTERFERENCES

No appeals or interferences known to Appellant, Appellant's legal representative, or assignee will directly affect or be directly affected by or have a bearing on the Board's decision in the present Appeal.

STATUS OF THE CLAIMS

Claims 1-23 were originally presented. Claims 1-2 were subsequently canceled, and Claims 5, 7-8, 12, 15-16 and 20-22 were amended, as noted in the Supplemental Amendment filed March 9, 2004 (the claims were misnumbered in the earlier filed Amendment A, dated February 3, 2004). Claims 3-23, which comprise all pending claims, stand finally rejected by the Examiner as noted in the Final Rejection dated May 24, 2004, and labeled Paper No. 10. The rejection of each pending claim is appealed.

STATUS OF AMENDMENTS

No amendment to the claims was proposed or entered subsequent to the Final Rejection dated May 24, 2004, and labeled Paper No. 10.

SUMMARY OF THE INVENTION

The present invention is directed to a processor and method of detecting and correcting data hazards within a processor, where a data hazard is defined at page 2, lines 17-19 of the present specification as a processor operating scenario "in which an out-of-order executed load instruction receives incorrect data."

As described in the present specification at page 2, line 21 *et seq.*, processors that support out-of-order execution of load instructions typically detect and correct data hazards by implementing a load queue that stores the target address of each load instruction that was executed out-of-order. Following execution of the out-of-order load instruction, addresses of exclusive transactions driven on the computer system interconnect by other processors, as well as

store instructions preceding the load instruction that are initiated by the processor itself, are snooped against the entries within the load queue. If a snooped exclusive transaction or a local store operation hits within the load queue, the entry is marked, for example, by setting a flag.

As discussed at page 2, line 36 *et seq.* of the present specification, when the processor thereafter executes a load instruction, the processor determines whether or not the load instruction precedes the flagged out-of-order load instruction in program order and whether or not the subsequently executed load instruction targets an address specified in a marked entry in the load queue. If so, a data hazard is detected, and the processor flushes and re-executes at least both load instructions, and possibly all instructions in flight following the first of the two load instruction in program order. The present invention recognizes that flushing and re-executing instructions in this manner to remedy data hazards results in a significant performance penalty, particularly for processors having wide instruction execution windows.

Figure 1 depicts a block diagram of an illustrative embodiment of a processor 10 having a reduced data hazard penalty in accordance with the present invention. As shown in Figure 1 and described at page 10, line 27 *et seq.*, processor 10 includes two load-store units (LSUs) 96 and 98 for executing load and store instructions, a store queue (STQ) 110 for managing store operations, and a load data queue (LDQ) 114 for managing load operations. As described at page 13, line 8 *et seq.*, LDQ 114 ensures that load data hazards are detected and appropriate remedial action is taken such that the later of two load instructions targeting the same address does not receive older data than the earlier of the two load instructions.

Figure 2 depicts an exemplary embodiment of LDQ 114 of processor 10. As illustrated in Figure 2 and described at page 13, line 19 *et seq.*, LDQ 114 includes a number of entries, each such entry including a effective address (EA) field 120 for storing the effective address (or address tag portion thereof) of a load instruction, a target address field 122 for storing the target address (or address tag portion thereof) from which the load instruction obtains data, a data field 124 for storing data loaded from memory by a load instruction, and a hazard field 126 for indicating that a hazard may exist for a load instruction. LDQ 114 is managed by queue management logic 128.

Figure 3A illustrates a flowchart of an exemplary method by which queue management logic 128 manages LDQ 114. As shown in Figure 3A and described at page 14, line 4 *et seq.*, the process begins at block 130 and then proceeds to block 132 in response to queue management logic 128 receiving a notification that a load instruction has been processed at some stage of the execution pipeline of processor 10. In response to this notification, queue management logic 128 determines at block 132 whether the load instruction has been dispatched, executed or completed by processor 10. In response to a determination that the load instruction has been dispatched, the process proceeds to block 134. Block 134 depicts queue management logic 126 allocating an entry in LDQ 114 for the newly dispatched instruction in accordance with the program order of the load instruction and placing the EA of the instruction within EA field 120. Thus, the location of an entry of a load instruction within LDQ 114 preferably indicates the program ordering of the load instruction with respect to other load instructions. Thereafter, the process returns to block 132.

As described at page 14, line 30 *et seq.* of the present specification, in response to a determination at block 132 that a load instruction has been completed, queue management logic 128 deallocates the entry corresponding to the completed load instruction, for example, by identifying an entry having a matching EA. Thereafter, the process returns to block 132.

If, on the other hand, queue management logic 128 determines from the received notification at block 132 that a load instruction has been executed by one of LSUs 96 and 98, the process proceeds to block 140, which, as described at page 15, line 11 *et seq.*, illustrates queue management logic 128 determining whether a later entry in LDQ 114 than the entry allocated to the executed load instruction has a target address in its target address field 122 that matches the target address of the executed load instruction.

If not, queue management logic 128 places the target address of the executed load instruction in the target address field 122 of the associated entry and places the data retrieved from memory (i.e., local cache, remote cache, or system memory) in response to execution of the load instruction in data field 124 of the associated LDQ entry, as shown at block 142. The entry

associated with the executed load instruction is also updated, as depicted at block 142, even if an entry associated with a later load instruction has a matching address if a determination is made at block 144 that hazard field 126 of the matching entry is not set. However, if hazard field 126 of the matching entry is set, a data hazard is detected.

As illustrated at block 146 and described at page 16, line 1 *et seq.*, to correct for the data hazard, queue management logic 128 places the target address for the executed load instruction in target address field 122 of the associated entry and utilizes the data contained in data field 124 of the matching entry of the later-in-program-order load to provide the data requested by the executed load instruction. That is, the data from data field 124 of the matching entry is provided to the general purpose register(s) specified by the executed load instruction and is also placed into data field 124 of the entry in LDQ 114 associated with the executed load instruction. The process returns to block 132.

As described at page 16, line 12 *et seq.*, the operation of queue management logic 128 thus minimizes the performance penalty associated with data hazards since the earlier-in-program-order load instruction need not be re-executed to obtain the correct data (i.e., in this case, the same data as the later-in-program-order load) and no flush of instructions is required.

Figures 4A-4C together illustrate an exemplary operating scenario in which a data hazard caused by a remote exclusive operation is detected and corrected in accordance with the present invention. As shown in Figure 4A and described at page 18, line 18 *et seq.*, when the operating scenario begins, two load instructions, which are designated LD1and LD2 in program order (with LD2 being the latest in program order), have been dispatched and accordingly have been allocated entries in LDQ 114 by queue management logic 128. In addition, LD2 has been executed out-of-order with respect to LD1, and the target address (TA) and data (D) have been loaded into the appropriate entry of LDQ 114 by queue management logic 128. The hazard field 126 of the entry associated with each of the load instructions is reset to 0.

Next, as shown in Figure 4B and as described at page 19, line 3 *et seq.*, in response to queue management logic 128 receiving notification of a remote request for exclusive access

having a target address that matches the TA of LD2, hazard field 126 of the entry associated with LD2 is set to 1. Then, as indicated in Figure 4C and described at page 19, line 8 *et seq.*, when LD1 is executed out-of-order and the execution generates a target address matching the TA specified in target address field 124 of the entry associated with LD2, a data hazard is detected. Accordingly, queue management logic 128 provides the data from data field 124 of the entry corresponding to LD2 to the register file to satisfy LD1 and also records the data in data field 124 and records the target address in target address field 122 of the entry corresponding to LD1. Thus, a data hazard caused by a remote exclusive operation intervening between out-of-order executed loads is detected and corrected without flushing or re-executing any instructions and without any additional latency.

ISSUES

(1) The non-statutory double patenting rejection of Claims 3 and 10 in view of commonly assigned U.S. Patent No. 6,725,358.

(2) The rejection of Claims 3-23 under 35 U.S.C. § 103 as unpatentable over U.S. Patent No. 5,913,048 to *Glew et al.* (*Glew*) in view of U.S. Patent No. 6,360,314 to *Webb, Jr. et al.* (*Webb*).

GROUPING OF THE CLAIMS

For purposes of this Appeal, Claims 3-23 stand or fall together as a single group.

ARGUMENT

I. Non-statutory double patenting rejection

In paragraph 3 of the first Office Action dated October 6, 2003, and labeled Paper No. 4, Claims 3 and 10 were subject to a non-statutory double patenting rejection in view of U.S. Patent Application Serial No. 09/598,435, which has now issued as U.S. Patent No. 6,725,358. In view of the issuance of the formerly co-pending application, Appellant submits a terminal disclaimer herewith to obviate the non-statutory double patenting rejection.

II. Rejection under 35 U.S.C. § 103

In paragraph 4 of the Final Rejection dated May 24, 2004, and labeled Paper No. 10, Claims 3-23 are rejected under 35 USC § 103(a) as unpatentable over U.S. Patent No. 5,913,048 to *Glew et al.* (*Glew*) in view of U.S. Patent No. 6,360,314 to *Webb, Jr. et al.* (*Webb*). That rejection is not well founded and should be reversed because the combination of *Glew* and *Webb* does not teach or suggest each feature recited in the present claims as required for a rejection under 35 U.S.C. § 103. MPEP 2143.

In particular, the combination of *Glew* and *Webb* does not teach or suggest, "queue management logic that, responsive to execution of a second load instruction, detects by reference to said load queue whether a data hazard exists, and if so, outputs said load data retrieved by said first load instruction from said entry to said register set in accordance with said second load instruction," as recited in exemplary Claim 1. In paragraph 5 of the Final Rejection, the Examiner relies upon *Glew's* memory order buffer (MOB) 150 as teaching the "queue management logic" recited in exemplary Claim 1, but notes in paragraph 6 of the Final Rejection that *Glew's* MOB 150 does not disclose the claimed technique of data hazard correction (viz. "outputs said load data retrieved by said first load instruction from said entry to said register set in accordance with said second load instruction"). Indeed, *Glew* explicitly teaches at col. 6, lines 63-65 that a data hazard occasioned by a remote processor writing a speculatively read location causes "the load and subsequent operation [to be] cleared and reexecuted to retrieve the correct data." Thus, *Glew* not only fails to disclose the claimed function of the "queue management logic" of Claim 1, but also explicitly teaches against Applicant's claimed technique of utilizing "load data retrieved by said first load instruction" and buffered within a load queue to satisfy the data transfer indicated by a second load instruction.

Because *Glew* does not disclose and, in fact, teaches against the recited operation of the "queue management logic" recited in Claim 1, the Examiner then cites col. 1, line 66 through col. 2, line 15, col. 4, lines 43-48, and col. 7, lines 61-63 of *Webb* as teaching the operation of the claimed "queue management logic." In relevant part, the cited passages teach that *Webb* solves the problem of a load instruction failing to load the data written by an earlier store instruction by providing:

> ... a bypass mechanism that compares the address of each load instruction with a set of recent stores that have not yet updated memory. A match of the recent stores provides the desired load data instead of having to retrieve the data from memory.

Webb, col. 4, lines 44-48. In other words, *Webb* teaches the use of store data from a store queue to satisfy a load operation indicated by a load instruction. Claim 1, in contrast, recites the use of load data within a load queue retrieved by a first load instruction to satisfy a load operation indicated by second load instruction. Clearly, the use of store data to correct an instruction ordering problem as taught by the Examiner's combination of *Webb* and *Glew* does not teach or suggest the use of load data to address a load data hazard as recited in exemplary Claim 1.

Furthermore, the "queue management logic" of exemplary Claim 1 would not have been obvious to those skilled in the art at the time of the present invention in view of the combination of *Glew* and *Webb*. Store queues, such as that disclosed by *Webb*, are required to buffer store data of a store operation until the indicated store is actually performed in memory (e.g., a data cache). It is therefore logical that *Webb* utilizes the stored data to satisfy a load operation, given the fact that the data in the store queue must be accessible and it is desirable, for correctness, that a load operation receives the newest data rather than the potentially stale data stored within the data cache.

However, this same reasoning is not applicable to the load queue and associated queue management logic recited in the present claims. Conventional load queues do not buffer load data and the contents of conventional load queues are not accessible by other instructions. Moreover, there is no teaching in any reference of record regarding the use of load data retrieved by one load instruction to satisfy another load instruction. Consequently, if a person of ordinary skill in the art at the time of the present invention, for example, an integrated circuit designer or architect, had before him the teachings of *Glew* and *Webb*, the present invention would not have been obvious because the cited references would offer no teaching or suggestion to modify the teachings of *Glew* and *Webb* with respect to a store queue and somehow apply those teachings to a load queue and associated queue management logic to obtain the invention recited in the present claims. This is particularly true given *Glew's* explicit teaching at col. 6, lines 63-65 that, in response to detection of a load data hazard, "the load and subsequent operations are cleared

[i.e., flushed] and reexecuted to retrieve correct data," rather than corrected by reference to a load queue, as claimed.

III. Conclusion

Because the combination of *Glew* and *Webb* does not teach or suggest, "queue management logic that, responsive to execution of a second load instruction, detects by reference to said load queue whether a data hazard exists, and if so, outputs said load data retrieved by said first load instruction from said entry to said register set in accordance with said second load instruction," the rejections of exemplary Claim 1, similar Claims 8 and 16, and their respective dependent claims under 35 U.S.C. § 103 should be reversed.

Please charge IBM Corporation's Deposit Account No. **09-0447** in the amount of $330.00 for submission of a Brief in Support of Appeal and $110.00 for the Terminal Disclaimer. No additional fee is believed to be required; however, in the event an additional fee is required please charge that fee to IBM Corporation's Deposit Account No. **09-0447**.

Respectfully submitted,

Brian F. Russell
Registration No. 40,796
DILLON & YUDELL LLP
8911 N. Capital of Texas Hwy., Ste. 2110
Austin, Texas 78759
(512) 343-6116

ATTORNEY FOR APPELLANTS